(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 707 265 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
17.04.1996 Bulletin 1996/16

(51) Int. Cl.[6]: **G06F 9/46**

(21) Application number: **95306349.2**

(22) Date of filing: **11.09.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **11.10.1994 US 320357**

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Cobb, Edward Ellis**
  Saratoga, California 95070 (US)
- **Freund, Thomas James**
  Austin, Texas 78759 (US)
- **Holdsworth, Simon Antony James**
  Andover, Hampshire, SP10 2NN (GB)
- **Houston, Iain Stuart Caldwell**
  Bradford Abbas, Sherborne DT9 6SD (GB)
- **Smith, Stanley Alan**
  Austin, Texas 78717 (US)

(74) Representative: **Moss, Robert Douglas**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) A system and method for creating an object oriented transaction service that interoperates with procedural transaction coordinators

(57) A system and method for efficiently employing procedural transaction managers from an object oriented transaction processing system. Implementation classes are introduced to bridge selected functions from an object oriented transaction processing system into a procedural system. Bridging allows both the reuse of existing procedural transaction managers and interoperation between procedural and object transactions systems which eases migration to new object oriented systems. Implementation classes include methods necessary to manage information necessary to use a procedural transaction API and to manage information returned by the procedural API.

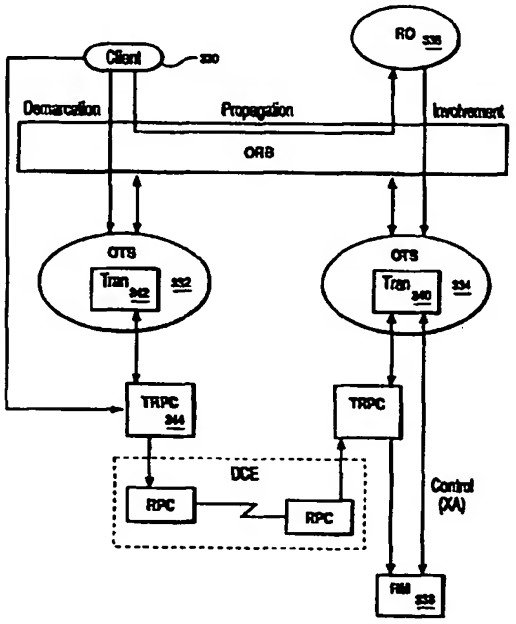


FIG. 5

EP 0 707 265 A2

Description

Field of the Invention

The present invention relates to transaction processing systems including stand-alone and distributed transaction processing systems. More particularly, the present invention relates to the use of computer implemented objects to implement a transaction processing system that supports object oriented transaction processing, but also integrates procedural transaction coordinators. Still more particularly, the present invention provides a system and method for allowing object oriented transaction processing applications to interoperate with procedural transaction processing applications.

Background and Related Art

Computer implemented transaction processing systems are used for critical business tasks in a number of industries. A transaction defines a single unit of work that must either be fully completed or fully purged without action. For example, in the case of a bank automated teller machine (ATM) from which a customer seeks to withdraw money, the actions of issuing the money, reducing the balance of money on hand in the machine and reducing the customer's bank balance must all occur or none of them must occur. Failure of one of the subordinate actions would lead to inconsistency between the records and actual occurrences.

Distributed transaction processing involves a transaction that affects resources at more than one physical or logical location. In the above example, an ATM transaction affects resources managed at the local ATM device as well as bank balances managed by a bank's main computer. A distributed transaction may not be physically distributed but may involve cooperating tasks that must be completed in synchrony for successful transaction completion.

The X/Open Company Limited (X/Open is a trademark of X/Open Company Ltd.) has promulgated a guide that describes one model for implementing distributed transaction processing. The X/Open Guide, Distributed Transaction Processing Reference Model, October, 1991, discusses the components of a distributed transaction system and the interrelationships between them. The X/Open Distributed Transaction Processing Model (the DTP Model) describes three main components: an Application Program (AP), a Transaction Manager (TM), and one or more Resource Managers (RMs). The Application Program uses and modifies the resources controlled by one or more of the Resource Managers. The Transaction Manager is responsible for global transactions and coordinates the decision whether to commit or roll-back the actions taken by the Resource Managers. (Commit causes the resources to be updated while roll-back causes all work to be discarded returning the resources to the state they were in upon transaction initiation.) The Resource Managers manage specific resources. Resource managers may include a database management system (DBMS), a file system, or similar resource.

Object oriented programming systems are designed to increase the efficiency of program development by enabling object reuse and simplifying system maintenance through clear separation of function. Each object in an object oriented system encapsulates the data for that object and the procedures or methods for operating on that data. Encapsulation means that the data for an object can be manipulated only by that object using the defined methods. Object oriented systems also implement object inheritance. Inheritance allows a more specific object to be derived from a general object. The more specific object can "inherit" all of the data and methods of the parent object, but can override selected data and methods and add others to implement its unique function.

The application of object oriented techniques to transaction processing systems raises many new issues but offers opportunities to increase system efficiency through the use of object oriented principles. The Object Management Group, Inc. (OMG) has established standards for interoperable object oriented systems. The overall architecture defined by OMG is the Common Object Request Broker Architecture (CORBA). CORBA defines the interactions between objects, and in particular, between distributed objects in different computer systems. OMG has accepted submission of a proposal to standardize transaction processing in object oriented systems. This submission, entitled the Object Transaction Service (OTS), sets forth the requirements for object services necessary to implement a transaction processing system. The OTS specification uses many of the unique capabilities of object oriented systems. The OTS model, however, is designed to allow interoperability with the X/Open DTP model and with existing procedural transaction processing systems through implementation of a mapping of the two-phase commit functions.

The proposed OMG Object Transaction Service describes mapping of the object oriented interfaces to existing X/Open DTP interfaces. This mapping is all that is specified in the OMG submission. The overall problem that exists is to define the methods required to give isolation to the application program interfaces and then to build the actual methods. The first problem within this overall problem is to define the methods necessary to allow object oriented transactional requests to interoperate with procedural transactional requests. These methods must allow for and support coordination of two-phase commit protocols between the OMG Object Transaction Service model and the X/Open DTP model. The mapping is between OMG functions (such as demarcation, propagation, involvement, and coordination) and procedural transaction manager functions (such as the formal two-phase commit protocols, transaction identifiers, directory/location

management, and transaction states). This coordination must occur within the transaction service and be isolated from the user interface level.

A second problem is provision of a mechanism to allow transactional object oriented application program requests to effectively exist with transactional procedural application program requests in a single atomic transaction. In particular, there is a need to develop an object based approach that provides the object-oriented application interfaces while also providing access to procedural transaction services without requiring changes to the procedural operations within the application program.

An additional problem is provision of an object based structure that is flexible enough to allow different procedural transaction managers to be plugged into the object oriented system without changing the overall object structure. Having this structure would allow the OMG Object Transaction Service interfaces to the client applications, Object Request Brokers, and resources (Resource Managers) to be preserved and also to preserve the classes that implement the function defined in the OMG OTS class specifications.

The technical problem therefore exists to develop a set of object oriented classes to efficiently connect object oriented applications to procedural transaction managers without modifying either the procedural transaction manager or existing procedural applications.

Summary of the Invention

The present invention is directed to a system and method for implementing an object transaction service that supports all OMG object oriented OTS application interfaces and provides transparent access to existing procedural transaction managers.

The changes required for differing procedural transaction managers are encapsulated within the implementation classes described in this invention. These implementation classes provide the bridge between the object environment and the existing procedural environments.

The present invention is directed to an object oriented computer system for transaction processing comprising: means for receiving a message to perform a transaction service operation; routing means for routing the message to an object method for performing the procedural transaction service; and implementation class means for enabling processing of the transaction service operation in response to the message. The system further includes means for receiving a result of the processing; and means for modifying object data based on the result.

The present invention is directed to an object structure that includes implementation classes to link the functions defined for the object oriented OMG OTS environment and existing procedural transaction managers running in existing environments. These classes provide a replaceable bridge between the OMG based classes and the actual procedural code running on a system. Implementation classes respond to messages generated by an OMG OTS compliant object and translate these messages into the procedural calls necessary to accomplish the required action using a procedural transaction manager as necessary. The implementation classes also receive results information from the procedural transaction manager and use that information to update OTS based objects in the system.

It is therefore an object of the present invention to provide a system that includes objects specifically structured and linked to bridge an object oriented transaction system to a procedural transaction system.

It is yet another object of the invention to provide a process for performing transactions using object oriented application program interfaces to affect an underlying procedural transaction manager.

It is yet another object of the invention to provide an object structure that allows different existing procedural transaction managers to be plugged in as the core transaction manager without requiring changes to the OTS classes that provide the OMG specified interfaces and behaviours.

Brief Description of the Drawing

The invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 is a block diagram illustrating the X/Open Distributed Transaction Processing Model.

Figure 2 is a block diagram illustrating the OMG Object Transaction Services model.

Figure 3 is a block diagram illustrating a system of distributed computers interconnected by networks in which the preferred embodiment of the present invention is applied.

Figure 4 is a block diagram of a computer system that implements the present invention.

Figure 5 is a diagram illustrating the transaction flow for interoperation between an object oriented and a procedural transaction environment.

Figure 6 is a diagram illustrating the relationship between object classes and procedural transaction services.

Figure 7 is a diagram illustrating the flow of control in a transaction according to the present invention.

Figure 8(a) and 8(b) are diagrams illustrating the process flow in a transaction according to the present invention.

Figure 9 is an example of the interface definition for a class according to the present invention.

Detailed Description

The X/Open Distributed Transaction Processing (DTP) model is shown generally in Fig. 1. An Application Program **102** executes and causes data or other resources to change state. Resources are managed by Resource Managers **106 108 110**, each of which can be a database management system (DBMS), file management system, communication Resource Managers (such as CPI-C, TxRPC, XATMI) or similar system. The Resource Managers may be distributed on computer systems remote from the system executing the Application Program **102** or they may be implemented as separate processes within the same computer system. Transaction Manager **104** controls the completion of processing for each particular transaction initiated by Application Program **102**. Transaction Manager **104** coordinates the actions of the Resource Managers to ensure that all resources are in a consistent state at the end of the transaction. This coordination ensures that the transaction appears to operate atomically, i.e. the transaction either changes all resources associated with the transaction or it changes none of them.

The Object Transaction Services model defined by the Object Management Group is shown generally in Fig. 2. A distributed client/server (C/S) application is shown at **120**. The application **120** comprises a number of objects that exchange messages to accomplish the actions required by the transaction. The objects present in the application include one or more Transactional Clients **122** that invoke operations of transactional objects. The object that begins a transaction is the transaction originator and the originator sends a message **138** to the Transactional Service at the beginning and end of a transaction. A transactional object is an object whose behaviour is affected by being invoked within the scope of a transaction. A transactional object typically contains or refers to persistent data that can be modified by transactional requests. Persistent data is that data that will survive a system restart. Persistent data typically resides on disk storage devices, non-volatile memory or similar devices.

Transactional objects are used to implement two types of application servers: a transactional server **124** and a recoverable server **126**. A recoverable server implements protocols necessary to respond to a transactional server and ensure that all participants in the transaction agree on the outcome, either to commit the transaction or roll-back the transaction, and to be able to recover from failure. A recoverable object is a transactional object, but not all transactional objects are recoverable. Non-recoverable transactional objects may implement their state using some other recoverable object.

A recoverable object must participate in Transaction Service **130** protocols. Transaction Services **130** maintain certain data defining the scope of each transaction as transaction context **132**. A transaction context **132** is associated with each ORB-aware thread (Object Request Broker (ORB) characteristics are defined by the OMG CORBA architecture.) The transaction context **132** is submitted with each request generated from the client application and is used to define operational environment characteristics where the request is processed. Contents of the transaction context **132** can include a reference to the transaction coordinator, ancestor references for nested transactions, a globally unique transaction id for the transaction coordinator and implementation specific data understood by the subordinate transaction coordinator.

Recoverable objects participate in Transactional Services **130** by registering a Resource **128** with the Transaction Service. The Transaction Service **130** drives the commit protocol (the two phase commit) by contacting those resources registered for a transaction.

A transactional server **124** is a collection of one or more objects whose behaviour is affected by the transaction but have no recoverable states of their own. A transactional server implements transactional changes using other recoverable objects. A transactional server does not participate in the completion of the transaction but can force the transaction to be rolled back by sending a roll back message **140**.

A recoverable server **126** is a collection of objects, at least one of which is recoverable. A recoverable server participates in the protocols by registering one or more Resource objects **128** with the Transaction Service using a Registration message **142**. The Transaction Service drives the commit protocol by issuing requests **144** to the resources registered for a transaction.

An example of a distributed processing system according to the present invention is shown generally in Fig. 3. Several computer systems are interconnecting using communication networks. For example, systems **212** and **204** are connected by network **210**. Systems **204, 202,** and **206** by network **208**. Systems **206, 216, 218, 220,** and **222** by network **214** and systems **222, 226,** and **228** by network **224**. The networks can be any known local area network (LAN) or wide

area network (WAN), including token ring, Ethernet or other network. The "network" can also be the communication bus between multiple processes in a single computer system.

A typical computer system is shown in Fig. 4. Each system **250** contains one or more central processing units **252**, volatile memory **254**, and input/output controller **256**. The input/output controller **256** manages writing to magnetic or optical disk storage **262**, removable storage **258**, **260** and to display **268**, keyboard **266** and pointing device **264**. System communication controller **270** manages communications with a network via communication link **272**. This configuration is provided for exemplary purposes only and is not intended to be limiting. A commercially available computer system such as the IBM PS/2 computer or IBM RISC System/6000 workstation are examples of the types of systems on which the invention may be practised. (PS/2 and RISC system/6000 are trademarks of the IBM Corporation.) As discussed above, the systems of a distributed environment may all be linked via a single communications bus sharing memory and disk storage.

Computer system **250** is controlled by an operating system such as the OS/2 operating system, or the AIX operating system (OS/2 and AIX are trademarks of the IBM Corporation.) Network communications may be managed by a network operating system such as Novell Netware operating system, or the IBM LAN Server operating system.

The present invention is practised using a program or suitable hardware to control a computer system such as those described above.

An object oriented application **120** performs transaction operations using the objects and classes defined by the OMG Object Transaction Services model. These classes provide an object oriented interface or API into the OMG OTS. The present invention solves the problem of efficiently accessing a procedural transaction manager by building an object transaction service around an existing procedural transaction coordinator.

This approach allows higher level classes to be used to provide a consistent object oriented interface to applications and to provide the capability to coordinate the overall atomic transaction. At the same time, it allows different underlying procedural transaction coordinators to be inserted into the final object transaction service. Examples of procedural transaction managers include Encina TRAN, the OS/400 Transaction manager, Tuxedo, TopEnd, CICS/ESA. The present invention can be implemented with any defined transaction manager and is not limited to those listed.

An example of interoperation according to the present invention is illustrated in Fig. 5. Fig. 5 illustrates interoperation between an OMG OTS and an Encina procedural transaction environment. The client application **330** invokes the object transaction services (OTS) through OMG defined object oriented interfaces. OTS **332** provides transaction services including propagating the transaction to the server using the OMG defined Object Request Broker (ORB) interfaces. Server OTS **334** involves a recoverable object **336** in the transaction as required and accesses an Encina procedural transaction manager (TRAN) **340** to synchronise transaction state information. As illustrated in the figure, the Encina procedural interface, TRAN, is imbedded in both the client and server OTS **340 342**. The details of how the procedural interface is embedded are set forth below. Within the client application **120**, procedural transaction requests (Encina requests in this example) can be made. The requests use the API provided through the TRAN remote procedure call interface (TRPC) **344**. A common TRAN **342** is accessed to manage the procedural transaction request which is sent to the TRAN **340** on the server. The request is processed by the existing Resource Manager (RM) **338**.

At points involving transaction state changes associated with the two-phase commit protocol, the procedural transaction managers (TRAN in this case **340 342**) synchronize the transaction to ensure atomic actions across both object and procedural resources involved in the transaction.

The present invention introduces a novel set of "implementation objects" to allow use of a procedural transaction manager from within the object oriented Object Transaction Services. The relationship of implementation classes to other transaction services is shown in Fig. 6. An object oriented application **120**, as discussed above, accesses OMG Object Transaction Services through an object oriented API **350**. This API is, in fact, the API specified by OMG for the Object Transaction Service and is provided and implemented by the OMG defined classes **352**. Implementation classes **354** are provided by the present invention to bridge OMG classes **352** to procedural transaction services **358**. The implementation classes **354** have defined object oriented interfaces **356** that are used by the OMG classes **352**. Implementation classes **354** communicate through a procedural API **360** to procedural transaction services **358**. This novel implementation preserves the standard OMG interface and, more importantly, the OMG class implementations. The OMG defined classes need not be modified to interact with different procedural transaction services. The implementation classes **354**, on the other hand, encapsulate the object to procedural interface in a generalized manner that can be readily adapted to different procedural transaction managers.

Objects in an object oriented software system perform operations on object data using object methods. An operation is requested by sending a message to a selected object requesting performance of a method. For example, the OMG OTS specifies that a new transaction is started by sending a *BEGIN()* message to the *CURRENT* object. (This is represented in shorter form as: *CURRENT::BEGIN().)* Any required parameters must be provided within the parentheses. This message will cause the *'BEGIN'* method of the object *'CURRENT'* to be invoked. The set of objects and methods defines the object oriented interface or API.

An object method can invoke other object methods or can carry out a function using functional code that is essentially procedural code. The flow of control in an object oriented system is illustrated in Fig. 7. A message **402** is sent by an

application program. The object oriented system routes **404** the message to an appropriate object for method selection using known object oriented system techniques. The requested method is selected **406** and executed **408**. The present invention includes the necessary procedural transaction statements in an object method. The procedural statement invokes a procedural API **410** requesting the necessary transaction processing. Results returned from the procedural call may be used to update the object's data **412** or may be included in a message **414** to another object requesting a change to its data or other action based on the results. This new message is routed using the message routing process **404**.

The ability of the present invention to issue procedural transaction API calls, to receive results from those calls, and to pass those results on to other objects in the system enable the object oriented OTS environment to be kept consistent even through it is using procedural transaction managers. The role of the transaction manager or transaction coordinator is to ensure that all resources under its control are managed as an atomic independent unit. The present invention performs tracking of resources by intercepting key procedural transaction functions and updating implementation class information based on these calls. An example of updated information is the registration of resources with the transaction coordinator.

The implementation classes of the present invention therefore transform the object oriented API defined by the OMG OTS specification into procedural actions that can be performed by a procedural transaction coordinator or transaction manager.

The OMG defined classes are reproduced in Table I below.

Table I

OMG Classes	
Class	Description
Current	Begin, end, and obtain information about a transaction
Factory	Create a new transaction along with necessary objects
Control	Obtains Terminator and Coordinator for a transaction
Terminator	Terminate by commit or rollback
Coordinator	Coordinate resources and two phase commit processing
RecoveryCoordinator	Coordinate recovery of a transaction
Resource	Represents the objects participating in the transaction
SubTransactionAware Resource	Subset of Resource required for nested transactions
Transactional Object	Used by object to indicate it is transactional

Implementation classes are constructed to isolate the common bridge points to procedural APIs. The preferred embodiment of the present invention includes a number of implementation classes, a subset of which are listed in Table II. Different numbers and types of classes may be employed without departing from the present invention. The number and type of implementation classes may vary depending on the identified bridge points to the procedural APIs.

Table II

Implementation Classes (Subset)	
Implementation Class	Description
SuperiorInfo	Collects information that maps to the superior Coordinator
NestingInfo	Tracks subordinate and ancestor Coordinators for nested transactions
RegisteredResources	Manage a set of Resource objects involved in a transaction
TransactionState	Maintain persistent state of a Coordinator
M_TransactionState	Metaclass of TransactionState
Transaction Manager	Manages the overall transaction using OMG and implementation objects

Each of the implementation classes can be described using a standard Interface Definition Language (IDL). The IDL describes the interface to a class including its methods and their arguments. An example of the TransactionManager class IDL is provided in Fig. 9.

An example of distributed transaction processing using the OMG and implementation classes is shown in Figs. 8(a) and 8(b). Fig. 8(a) illustrates client side processing where client **500** sends a request to an object **502**. The Transaction Service utilizes OMG and implementation classes such as the local instance of the TransactionManager **504** to prepare and track the request which is sent to the server via ORB request **506**.

Fig. 8(b) illustrates the server side processing. The request is received by the Transaction Service and tracked by a TransactionManager instance **508** using an instance **510** of class Coordinator **512** and instances **514 516 518 520** of the implementation classes SuperiorInfo, NestingInfo, RegisteredResources, and TransactionState. The request **522** is passed to a TransactionObject **524** for completion of request processing. The reply flows back through the ORB to the client object.

As previously mentioned, the implementation classes provide the bridge between the object transaction service operations and the procedural transaction service. The implementation classes encapsulate the procedural API usage so that changing the underlying procedural transaction manager is transparent to the OMG classes. As the different procedural transaction services are integrated into the OTS, the areas where the functions must bridge to each other must be determined. This evaluation will determine which of the implementation classes must be changed to access or coordinate with procedural transaction manager functions. The implementation class changes will involve updating the procedural code within the class to use resources associated with the procedural transaction service. Key areas where this will be done include transaction state changes associated with beginning and ending transactions. Procedural function calls to request distribution of *prepare*, *commit*, and *rollback* requests will be changed within the implementation class.

It was also mentioned previously that usage of a common procedural transaction manager aids implementation of interoperability of object oriented and procedural transaction requests. The implementation classes are where this is coordinated to ensure atomic actions across all resources associated with the overall transaction. The implementation classes involved with transaction state changes include TransactionState and TransactionManager. (See for example, instances **510** and **508** in Fig. 8(b).) These classes must be changed to receive and initiate the API requests for beginning and ending transactions. Coordination within the implementation classes of a common procedural transaction manager for both the Object Transaction Service and the Procedural Transaction Service gives the capability to have an atomic transaction that avoids the traditional gateways between transaction managers. It also preserves the integrity of the application program because it does not require addition of coordination code within the program and it does not change existing APIs.

Claims

1. A computer implemented process for operating a procedural transaction manager from an object oriented transaction processing system, the process comprising the steps of:
   receiving an object based message to perform a transaction service from a service requester;
   routing said object based message to a first object from a first set of objects having a transaction method corresponding to said transaction service;
   generating an object based message to perform one or more tasks of said transaction service;
   routing said one or more object based messages to one or more objects from a second set of objects having a task transaction method corresponding to said one or more tasks; and
   executing a procedural transaction service from within said task transaction method.

2. The process of claim 1, further comprising the steps of:
   receiving a result from said procedural transaction service; and
   transmitting an object based message with the results of said procedural transaction to said service requester.

3. The process of claim 1, wherein said first set of objects conform to the Object Management Group Object Transaction Services specification.

4. The process of claim 2, wherein said first set of objects conform to the Object Management Group Object Transaction Services specification.

5. The process of claim 2, wherein said procedural transaction service is an Encina transaction processing system.

6. An object oriented system for transaction processing, the system comprising:
   means for receiving a message to perform a procedural transaction service;

routing means for routing said message to an object method for performing said procedural transaction service; and

implementation class means for enabling processing of said procedural transaction service in response to said message.

7. The system of claim 6, wherein said implementation class means includes means for receiving a result of said processing; and wherein the system further comprises:

means for modifying object data based on said result.

8. The system of claim 6, wherein the object methods conform to the Object Management Group Object Transaction Services specification.

9. The system of claim 6, wherein said procedural transaction service includes Encina transaction processing services.

10. The system of claim 6, wherein said implementation classes include:

a TransactionManager class for managing the overall transaction using OMG Object Transaction Services and implementation objects;

a TransactionState class for managing the persistent state of a coordinator;

a SuperiorInfo class for collecting information that maps to a superior coordinator; and

a RegisteredResources class for managing a set of resource objects involved in a transaction.

11. The system of claim 7, wherein said implementation classes include:

a TransactionManager class for managing the overall transaction using OMG Object Transaction Services and implementation objects;

a TransactionState class for managing the persistent state of a coordinator;

a SuperiorInfo class for collecting information that maps to a superior coordinator; and

a RegisteredResource class for managing a set of resource objects involved in a transaction.

12. The system of claim 6, wherein said routing means includes communication means for routing said messages between distributed objects.

13. The system of claim 11, wherein said communication means conforms to the Object Management Group Object Request Broker specification.

14. A distributed transaction processing system, the system comprising:

client means for receiving a transaction request;

testing means for determining a transaction manager type for processing said transaction request;

transaction manager selection means for invoking one of a plurality of transaction managers in response to the testing means;

implementation class means for bridging information between an object oriented transaction environment and a procedural transaction environment; and

a plurality of communication means for communicating with a transaction processing server in response to a request from said selected transaction manager.

15. The system of claim 14, wherein said client means conforms to the Object Management Group Object Transaction Services specification.

16. The system of claim 14, wherein said plurality of communications means includes OSF Distributed Communications Environment protocols and OMG Object Request Broker protocols.

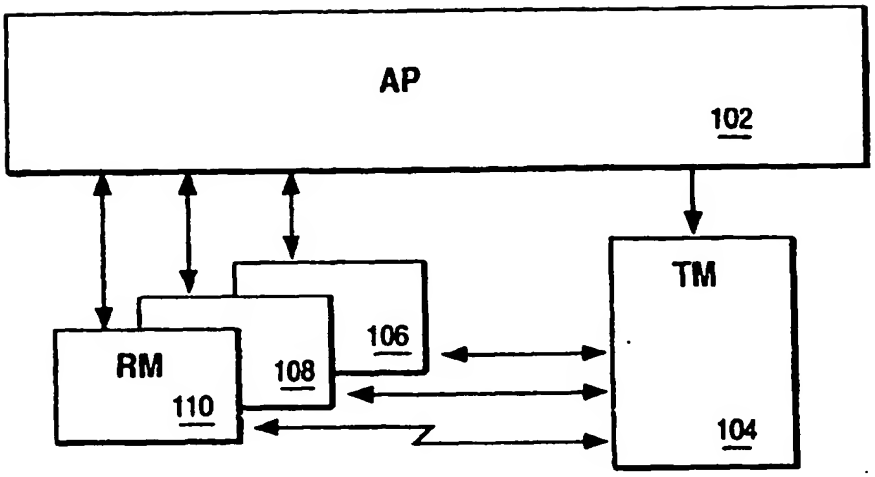
17. The system of claim 14, wherein said implementation class means include:

TransactionManager class means for managing the overall transaction using OMG Object Transaction Services and implementation objects;

TransactionState class means for managing the persistent state of a coordinator;

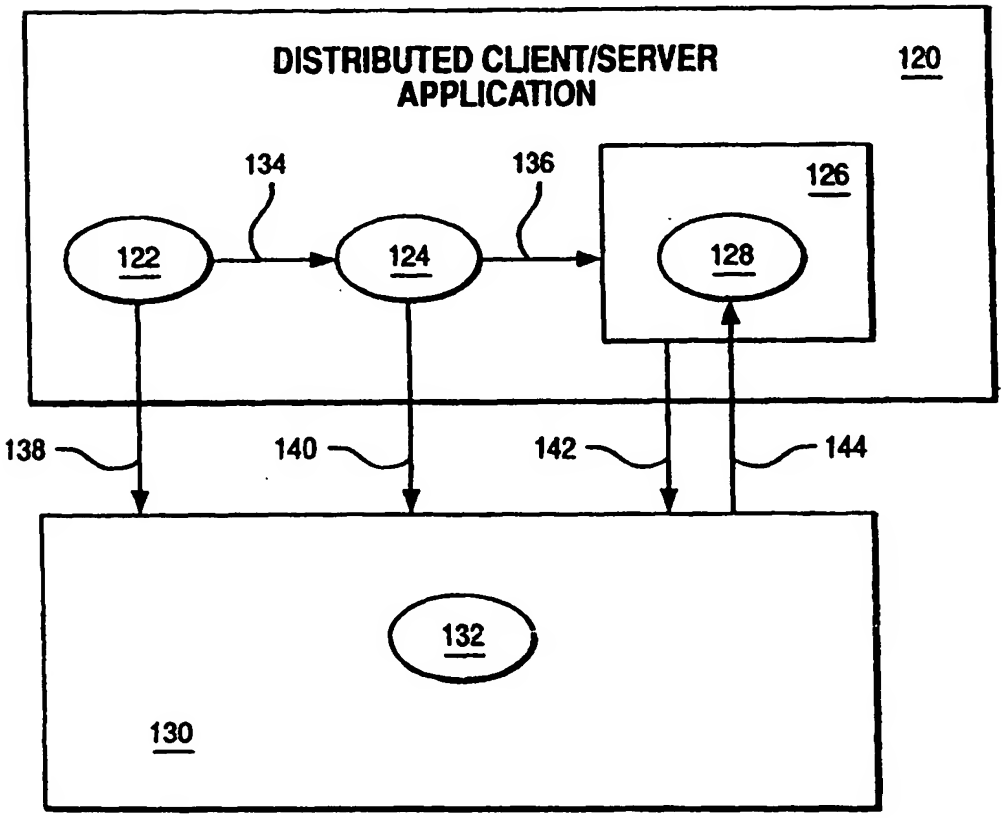
SuperiorInfo class means for collecting information that maps to a superior coordinator; and

RegisteredResource class means for managing a set of resource objects involved in a transaction.



PRIOR ART

FIG. 1



PRIOR ART

FIG. 2

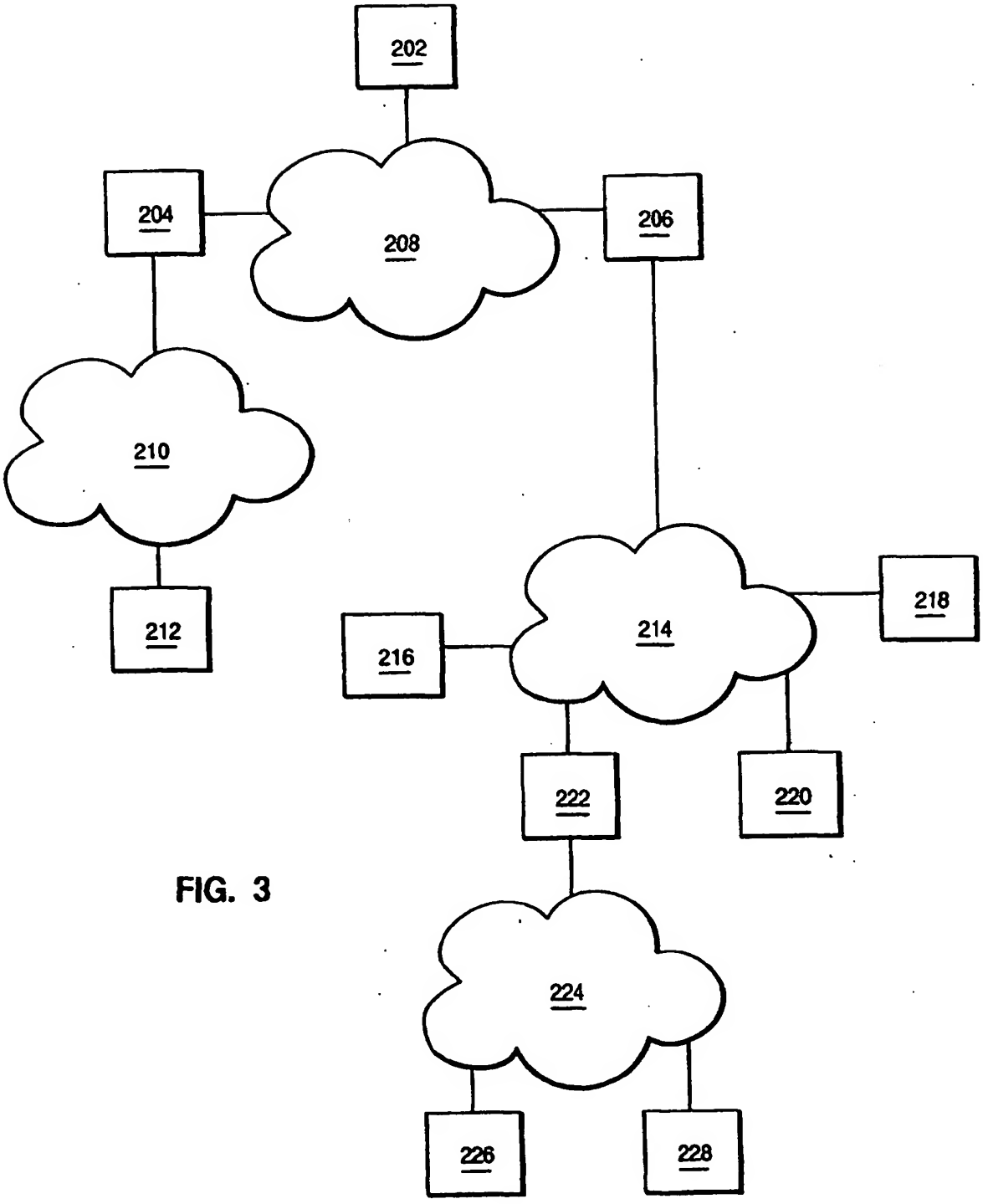


FIG. 3

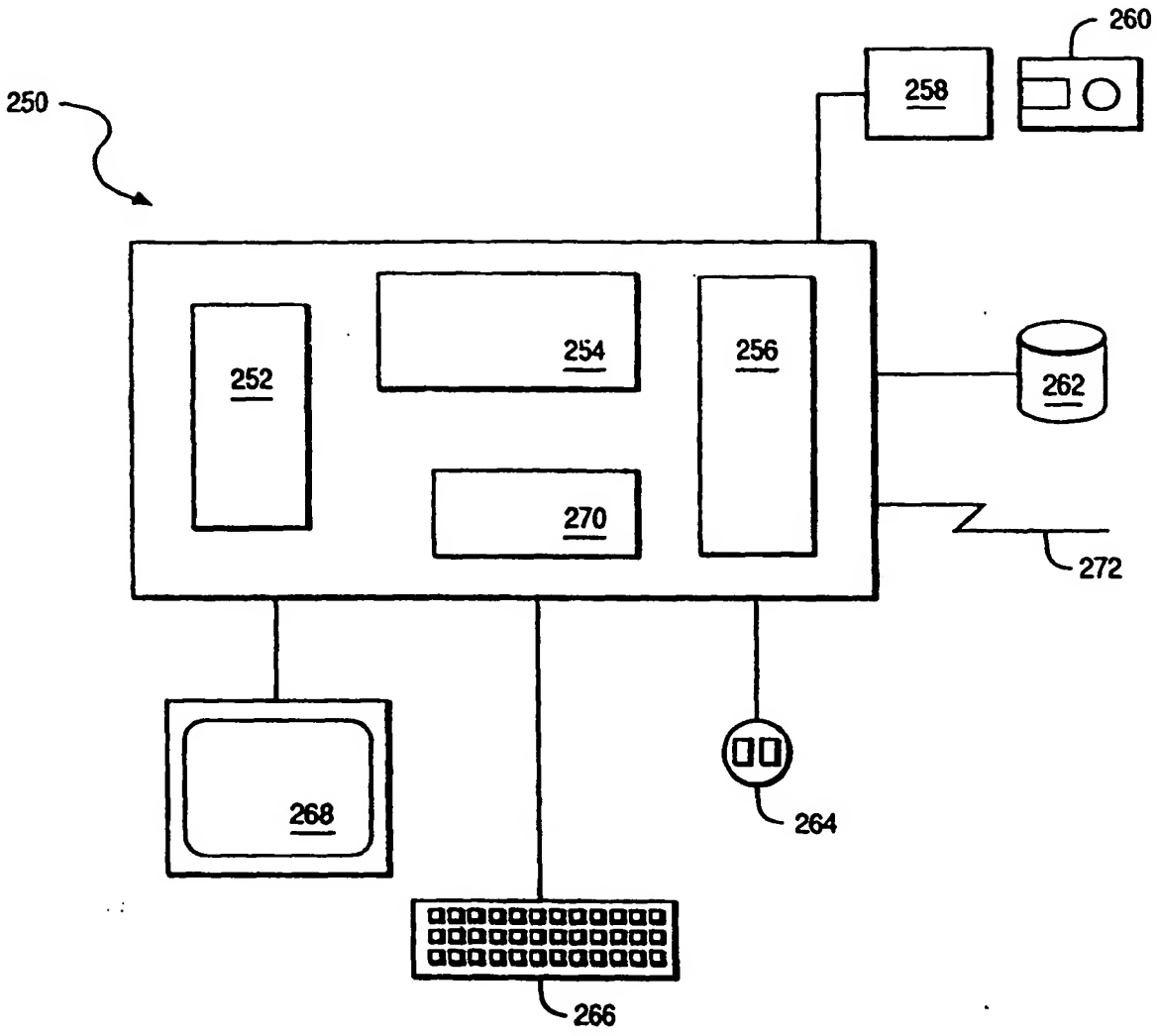


FIG. 4

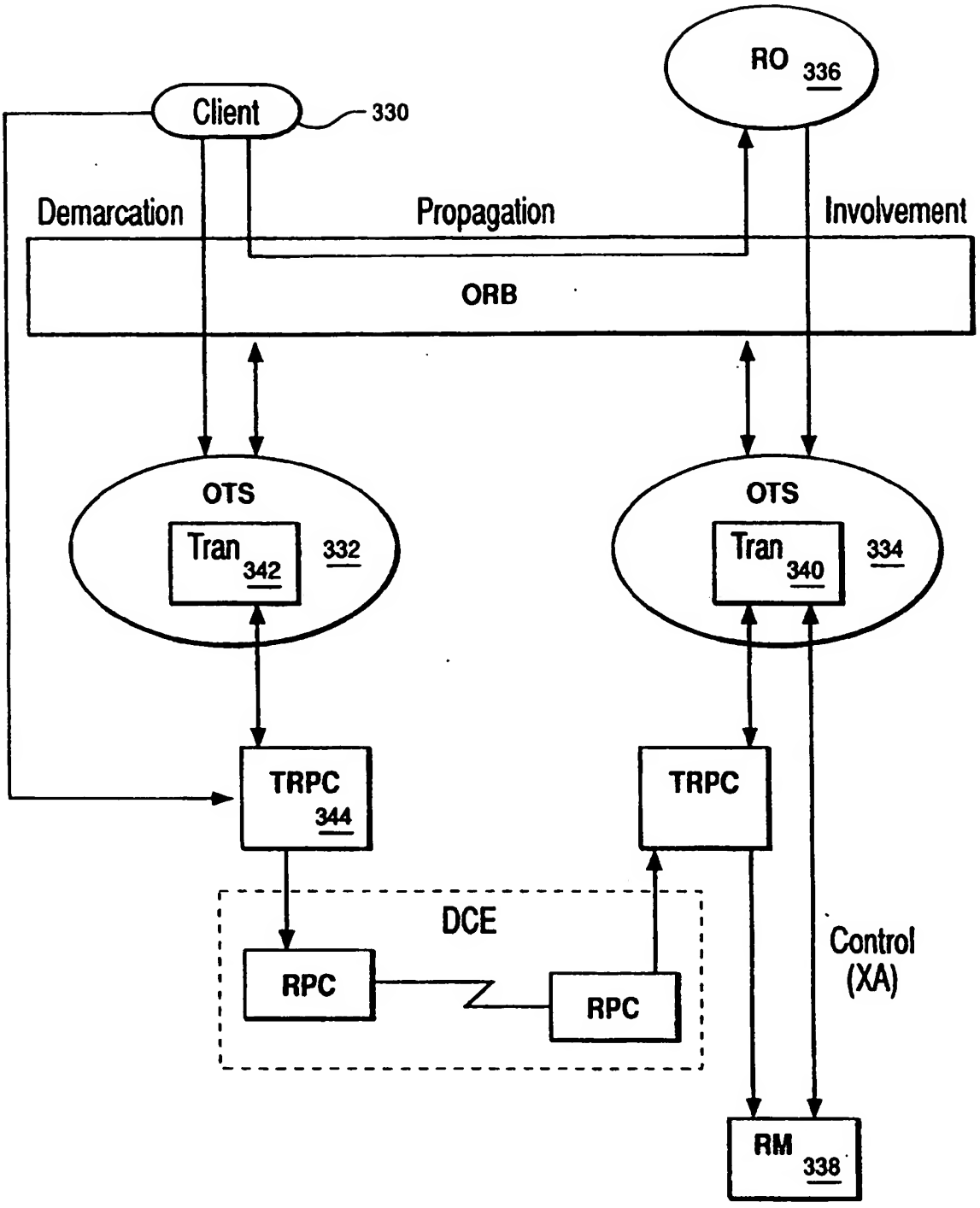

RO 336
Client
330
Demarcation
Propagation
Involvement
ORB
OTS
Tran 342
332
OTS
Tran 340
334
TRPC 344
TRPC
DCE
RPC
RPC
Control (XA)
RM 338

FIG. 5

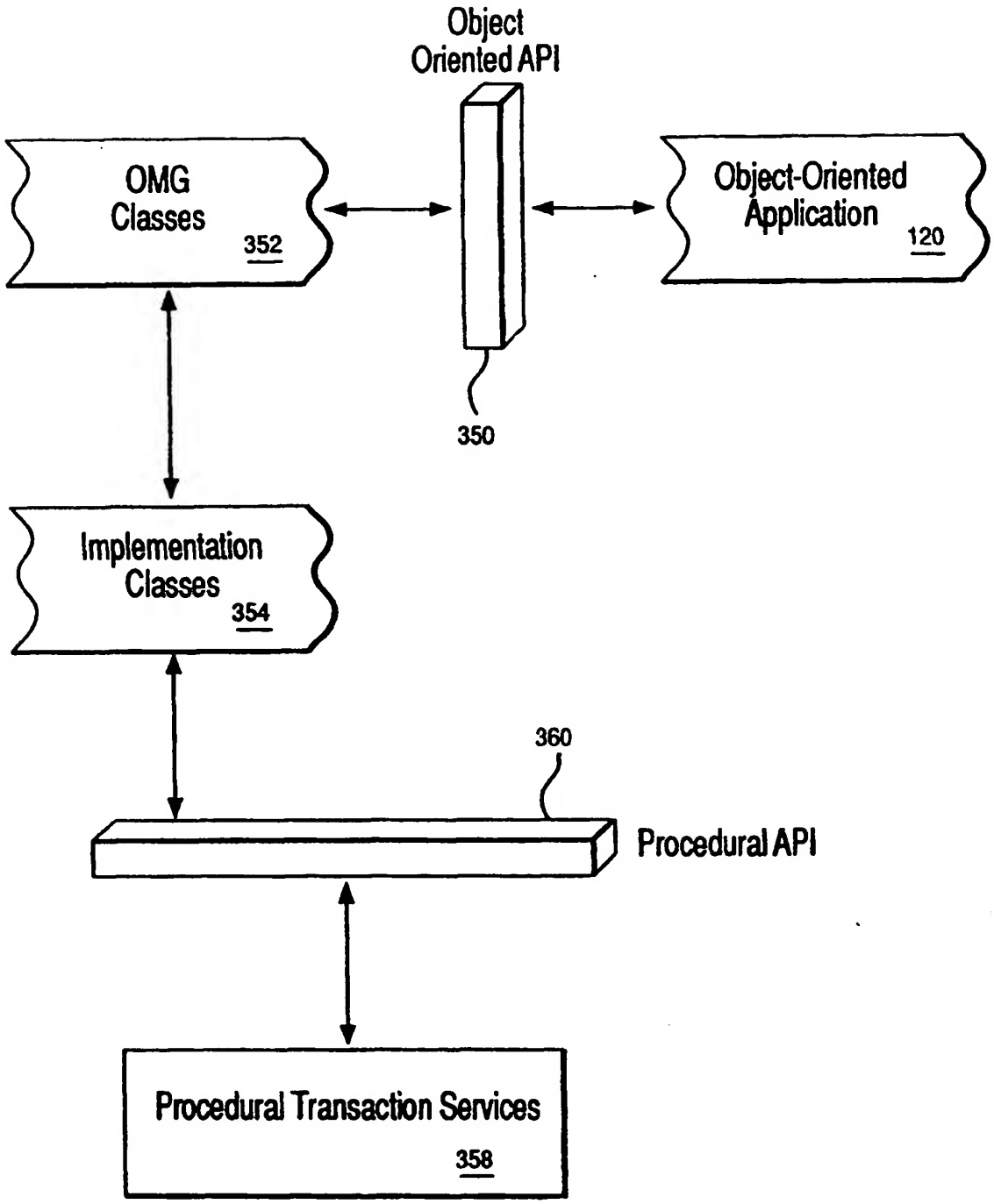

Object
Oriented API
OMG
Classes
352
Object-Oriented
Application
120
350
Implementation
Classes
354
360
Procedural API
Procedural Transaction Services
358

FIG. 6

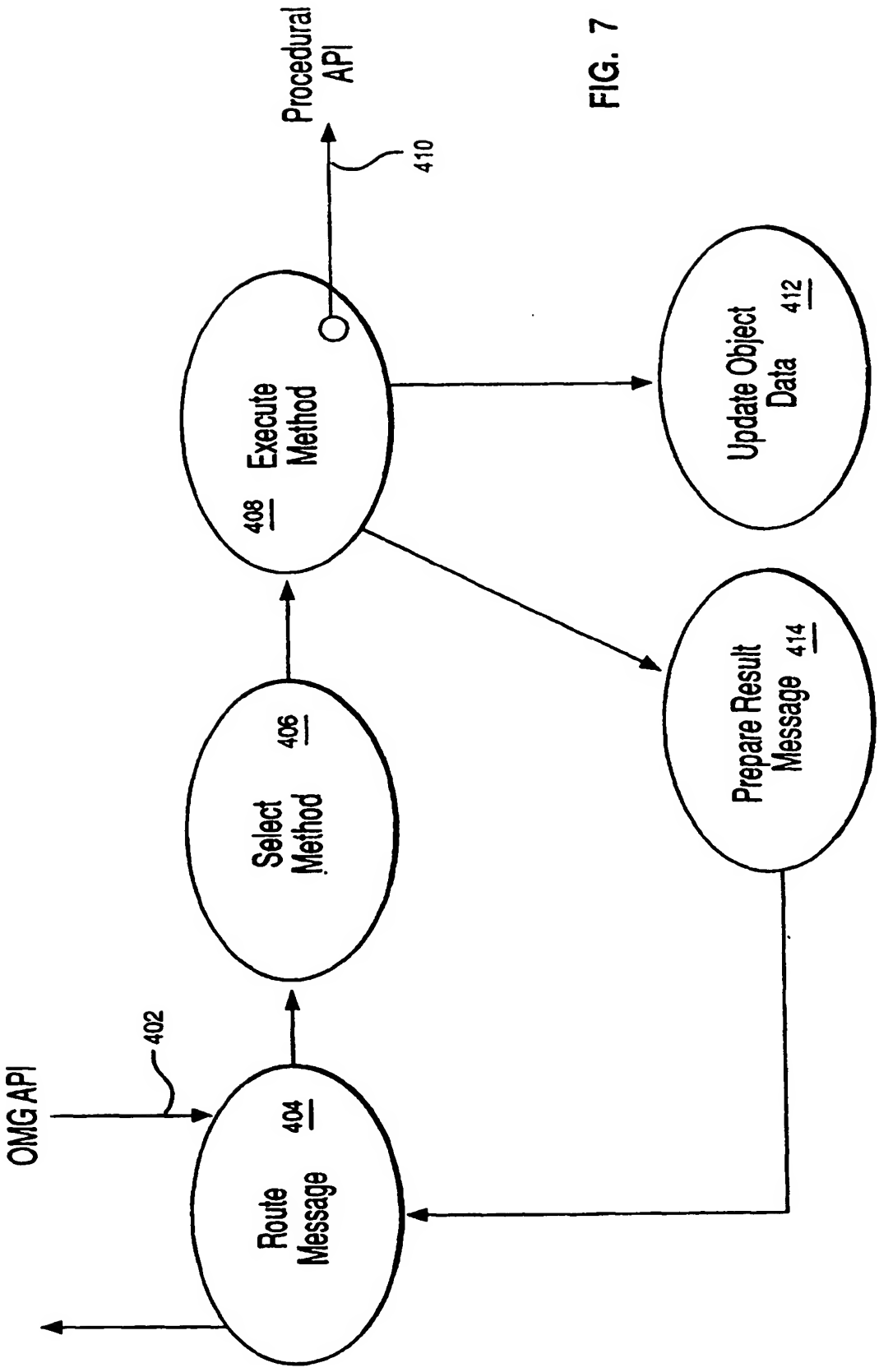
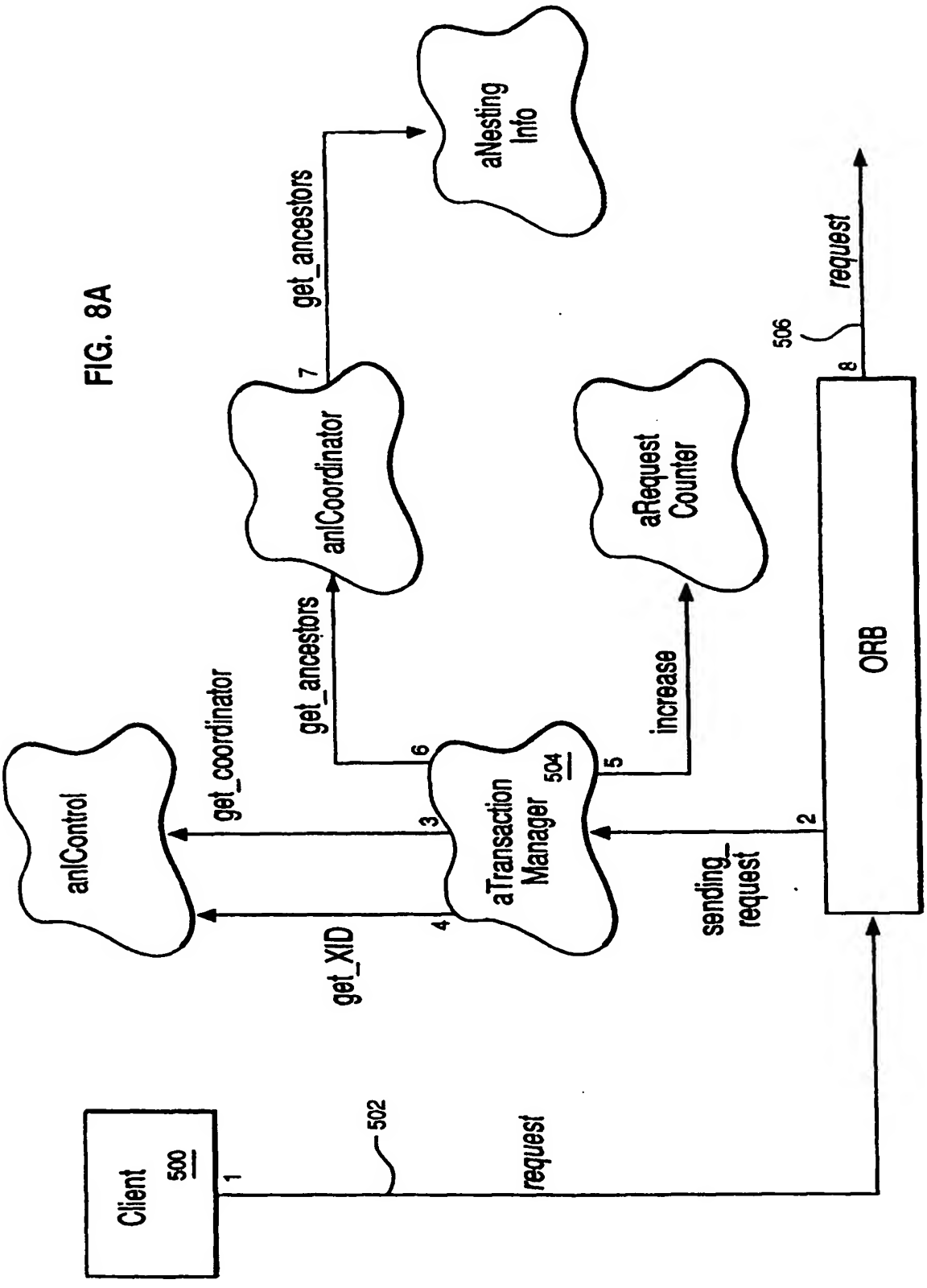

OMG API
402
Route Message 404
Select Method 406
Execute Method 408
Procedural API
410
Update Object Data 412
Prepare Result Message 414

FIG. 7

FIG. 8A


Client
500
1
502
request
anControl
get_coordinator
3
get_XID
4
aTransaction Manager
504
6
get_ancestors
anCoordinator
7
get_ancestors
aNesting Info
5
increase
aRequest Counter
sending_ request
2
ORB
8
506
request

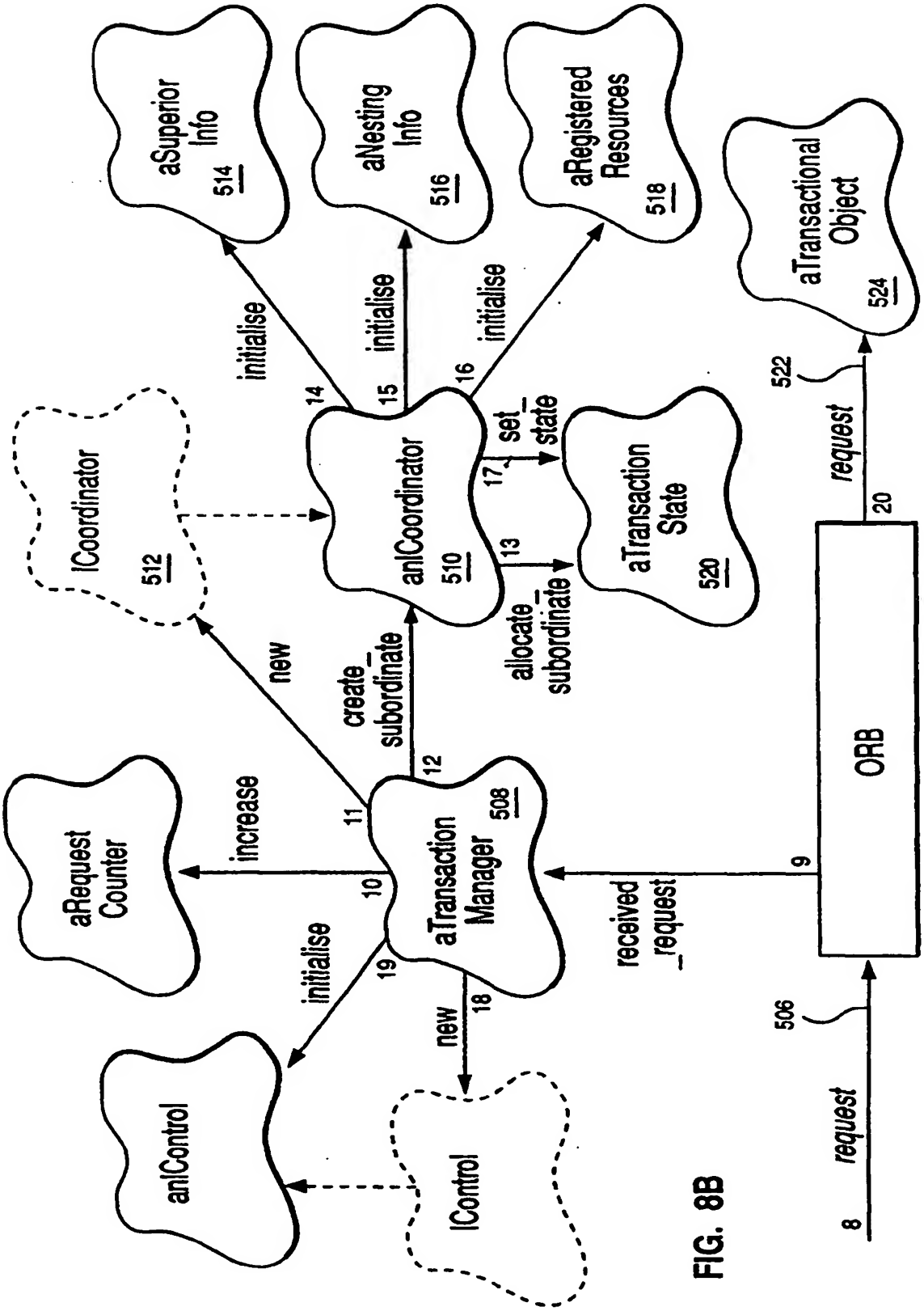

aSuperior Info
514
aNesting Info
516
aRegistered Resources
518
aTransactional Object
524
ICoordinator
512
anICoordinator
510
aTransaction State
520
aRequest Counter
aTransaction Manager
508
ORB
anIControl
IControl
initialise
initialise
initialise
new
create_ subordinate
set_ state
allocate_ subordinate
increase
initialise
new
received _request
request
request
8
506
9
10
11
12
13
14
15
16
17
18
19
20
22

FIG. 8B

Transaction Manager Service

```
interface TransactionManager : Cooperation
  {                                    // metaclass: SingleInstance
//# Update operations

  boolean add_coordinator(   in XID global_id
                             in Coordinator coord,
                             in unsigned long timeout  ),

  boolean set_current(   in IControl tc,
                         in boolean stack  );
  IControl end_current(  in boolean unstack  );
  boolean remove_coordinator(   in XID tid  );

//# Query operations

  IControl        get_current( );
  ICoordinator    get_current_coordinator( );
  ICoordinator    get_coordinator( in XID global_id );
  unsigned long   num_active( in XID global_id ,
                              out boolean outstanding  );
  }
```

FIG. 9

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 707 265 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
29.10.1997 Bulletin 1997/44

(43) Date of publication A2:
17.04.1996 Bulletin 1996/16

(21) Application number: **95306349.2**

(22) Date of filing: **11.09.1995**

(51) Int. Cl.[6]: **G06F 9/46**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **11.10.1994 US 320357**

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Cobb, Edward Ellis**
  Saratoga, California 95070 (US)
- **Freund, Thomas James**
  Austin, Texas 78759 (US)
- **Holdsworth, Simon Antony James**
  Andover, Hampshire, SP10 2NN (GB)
- **Houston, Iain Stuart Caldwell**
  Bradford Abbas, Sherborne DT9 6SD (GB)
- **Smith, Stanley Alan**
  Austin, Texas 78717 (US)

(74) Representative: **Moss, Robert Douglas**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) A system and method for creating an object oriented transaction service that interoperates with procedural transaction coordinators

(57) A system and method for efficiently employing procedural transaction managers from an object oriented transaction processing system. Implementation classes are introduced to bridge selected functions from an object oriented transaction processing system into a procedural system. Bridging allows both the reuse of existing procedural transaction managers and interoperation between procedural and object transactions systems which eases migration to new object oriented systems. Implementation classes include methods necessary to manage information necessary to use a procedural transaction API and to manage information returned by the procedural API.

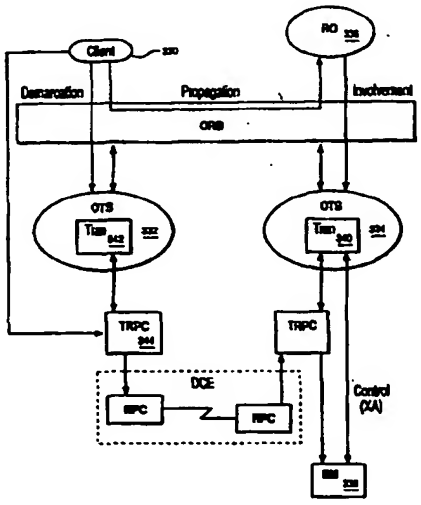


FIG. 5

EP 0 707 265 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 6349

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, ARLINGTON, TEXAS, MAY 20 - 24, 1991, no. CONF. 11, 20 May 1991, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, pages 65-72, XP000221844 POPOVICH S S ET AL: "AN OBJECT-BASED APPROACH TO IMPLEMENTING DISTRIBUTED CONCURRENCY CONTROL" * page 66, right-hand column, line 42 - page 68, left-hand column, line 31 * * page 71, left-hand column, line 1 - right-hand column, line 8 *	1-17	G06F9/46
X	EP 0 613 083 A (SUN MICROSYSTEMS INC) 31 August 1994 * page 3, line 45 - page 4, line 32 * * page 7, line 1 - page 9, line 31 *	1-17	
A	IEEE SOFTWARE, JAN. 1991, USA, vol. 8, no. 1, ISSN 0740-7459, pages 66-73, XP002037836 SHRIVASTAVA S K ET AL: "An overview of the Arjuna distributed programming system" * the whole document *	1-17	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 August 1997	Brandt, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)